INTERFACE TO A PROGRAMMABLE LOGIC CONTROLLER

DESCRIPTION

Related Applications

This application is related to a co-pending U.S. Patent Application entitled "Embedded File System for a Programmable Logic Controller," filed on December 30, 1998, and having the same common assignee.

Technical Field

The present invention relates to industrial automation, and in particular to an interface for accessing, controlling and monitoring a programable logic controller.

Background of the Invention

Sophisticated industrial processes require the cooperative execution of numerous interdependent tasks by many different pieces of equipment. The complexity of ensuring proper task sequencing and management requires not only procedural logic, but also constant monitoring of equipment states to organize and distribute operations and detect malfunctions.

Today, many industries use programmable logic controllers to properly operate and monitor elaborate industrial equipment and processes. Programmable logic controllers operate in accordance with a stored control program that causes the controller to examine the state of the controlled machinery by evaluating signals from one or more sensing devices (e.g., temperature or pressure sensors), and to operate the machinery (e.g., by energizing or de-energizing operative components) based on a procedural framework, the sensor signals and, if necessary, more complex processing.

Ordinarily, process operation is monitored, at least intermittently, by supervisory personnel by means of one or more central management stations. Each station samples the status of controllers (and their associated sensors) selected by the operator and presents the data in some meaningful format. The

management station may or may not be located on the same site as the monitored equipment; frequently, one cental station has access to multiple sites (whether or not these perform related processes). Accordingly, communication linkage can be vital even in traditional industrial environment where process equipment is physically proximate, since at least some supervisory personnel may not be.

To facilitate the necessary communication, the controller and related computers (such as monitoring stations) are arranged as a computer network that uses some consistent protocol to communicate with one another. The communication protocol provides the mechanism by decomposing and routing messages to a destination computer identified by an address. The protocol may place a "header" of routing information on each component of a message that specifies source and destination addresses, and identifies the component to facilitate later reconstruction of the entire message by the destination computer. This approach to data transfer permits the network to rapidly and efficiently handle large communication volumes without reducing transfer speed in order to accommodate long individual message.

In typical computer networks having one or more programable logic controllers, a monitoring computer, which may be remotely located from any or all of the controllers to which it has access, periodically queries the controllers to obtain data descriptive of the controlled process or machine, or the controller itself. This data is then available for analysis by the monitoring computer.

Today, the programs to access, control, and monitor programable logic controllers are written in programing languages that are not easily customizable by a user.

Summary of the Invention

The present invention provides an interface for accessing, controlling and monitoring a programable logic controller with a network client having a conventional web browser.

The present invention includes an archive, an interface, and a library. The archive provides for compiling an application written in code supported by the web browser. The interface is responsive to the application for establishing a connection between the controller and the network client. Further, the library is responsive to the application for supporting communication between the controller and the network client upon establishing a connection therebetween.

Brief Description of the Drawings

FIGURE 1 is a simplified block diagram of a server operably connected between a programable logic controller and a host computer;

FIGURE 2 is a simplified block diagram of an interface in accordance with the present invention having a communications library and operably connected between the server and the host computer of FIGURE 1; and

FIGURE 3 is a simplified block diagram of the communications library of FIGURE 2.

Detailed Description

The Internet is a worldwide "network of networks" that links millions of computers through tens of thousands of separate (but interconnecting) networks. Via the Internet, users can access tremendous amounts of stored information and establish communication linkages to other Internet-based computers.

Much of the Internet is based on the client-server model of information exchange. This computer architecture, developed specifically to accommodate the "distributed computing" environment characterizing the Internet and its component networks, contemplates a server that services requests of other computers or clients

that connect to it. The clients usually communicate with a single server or can use the server to reach other servers.

To ensure proper routing of messages between the server and the intended client, the messages are first broken up into data packets, each of which receives a destination address according to a consistent protocol, and which are reassembled upon receipt by the target computer. A commonly accepted set of protocols for this purpose are the Internet Protocol, or IP, which dictates routing information; and the transmission control protocol, or TCP, according to which messages are actually broken up into IP packets for transmission for subsequent collection and reassembly. TCP/IP connections are quite commonly employed to move data across telephone lines.

The Internet supports a large variety of information-transfer protocols such as the World Wide Web (hereinafter, simply, the "web"). Web-accessible information is identified by a uniform resource locator or "URL," which specifies the location of the file in terms of a specific computer and a location on that computer. Any Internet "node"-that is, a computer with an IP address (e.g., a server permanently and continuously connected to the Internet, or a client that has connected to a server and received a temporary IP address)-can access the file by invoking the proper communication protocol and specifying the URL. Typically, a URL has the format http://<host>/<path>, where "http" refers to the HyperText Transfer Protocol, "host" is the server's Internet identifier, and the "path" specifies the location of the file within the server. Each "web site" can make available one or more web "pages" or documents, which are formatted, tree-structured repositories of information, such as text, images, sounds and animations.

An important feature of the web is the ability to connect one document to many other documents using "hypertext" links. A link appears unobtrusively as an underlined portion of text in a document; when the viewer of this document moves

the cursor over the underlined text and clicks, the link - which is otherwise invisible to the user - is executed and the linked document retrieved. That document need not be located on the same server as the original document.

Hypertext and searching functionality on the web is typically implemented on the client machine, using a computer program called a "web browser." With the client connected as an Internet node, the browser utilizes URLs- provided either by the user or the link- to locate, fetch and display the specified documents. "Display"in this sense can range from simple pictorial and textual rending to realtime playing of audio and/or video segments or alarms, mechanical indications, printing, or storage of data for subsequent display. The browser passes the URL to a protocol handler on the associated server, which then retrieves the information and sends it to the browser for display; the browser causes the information to be cached (usually on a hard disk) on the client machine. The web page itself contains information specifying the specific Internet transfer routine necessary to retrieve the document from the server on which it is resident. Thus, clients at various locations can view web pages by downloading replicas of the web pages, via browsers, from servers on which these web pages are stored. Browsers also allow users to download and store the displayed date locally on the client machine.

Most web pages are written in HyperText Markup Language, or HTML, which breaks the document into syntactic portions (such as headings, paragraphs, lists, etc.) that specify layout and contents. An HTML file can contain elements such as text, graphics, tables and buttons, each identified by a "tag." Markup languages, however, produce static web pages.

However, if desired, web-page designers can overcome the static page appearance dictated by HTML. The Java language is a well-known, machine-independent, interpreted computer language that facilitates dynamic display of information. Java-encoded "applets" are stand-alone programs embedded within

web pages that can interact with the user locally, display moving animations and perform other functions on "Java-capable" browsers- that is, browsers which include a Java interpreter. The applet is transferred to the browser along with other web-page information and is executed by the Java interpreter; the data acted upon by the applet can be located on the same or a different web page, or a different server entirely, since applets can themselves cause the browser to retrieve information via hypertext links.

For example, suppose that a client users instructs the client-resident browser to obtain a document having the URL http://host/file.html. The browser contacts the HTTP server running on "host," and requests the document file.html. The server finds this document and sends it according to the proper Internet protocol, along with a Multipurpose Internet Mail Extension or "MIME" identifier that specifies the document's type. When the client receives the document, the browser examines the MIME to determine whether it is capable of autonomously displaying the document, or whether an external resource (e.g., a specialized viewer to display video segments) is necessary. In a simple case, the document might contain text and graphics specified in HTML, and specify and image residing in a different file on a different server or on the same server. The browser renders the document in accordance with the HTML instructions and requests the image, displaying it in the document as specified by the instructions when the image arrives. In more complex cases the document may contain, for example, Java instructions, which are passed to the browser's Java interpreter.

Key to the concept of a web page, therefore, is the division of functionality between the client-based browser and the server-based web page, and the particular roles assigned to each. The browser locates, fetches and displays resources, executes hyperlinks and applets, and generally interprets web-page information; the web page contains data, hyperlink addresses, transfer protocols and computer

instructions defining "potential functionality" that may be executed by the browser. Ordinarily, web pages reside on servers accessible via the Internet. However, the above-discussed mode of splitting functions between web pages and browsers can be instituted on internal networks as well. These networks, sometimes called "intranets," support the TCP/IP communication protocol and typically serve the needs of a single business (of business department), which may be located at a single site (with individual clients connected by a simple local-area network) or multiple physically dispersed sites requiring a wide-area network. Various of the computers forming the intranet network can be utilized as servers for web pages, each with its own URL and offering access to network client computers via TCP/IP.

A server 12 operably connected between a programable logic controller 14 and a host computer 16 is depicted in FIGURE 1. The server 12 provides web access to controller data (i.e., variable, system diagnostics, configuration information, I/O status) through "thin clients" (i.e., web browsers). The server 12 provides meaningful, structured information to users via the open communication standard to TCP/IP and HTTP. In particular, anyone with a web browser can browse the controller as if it was just another web site.

Examples of other web interfaces to a programmable controller are disclosed in U.S. Application Serial No. 08/927,005, filed on September 10, 1997, which is incorporated herein by reference.

In FIGURE 1, the programable logic controller 14 is conventional and includes one or more storage devices indicated generally at 18. The controller 14 includes a CPU module 19 for executing program instructions retrieved from storage 18 to operate, for example, a piece of industrial equipment. The storage device 18 typically is composed of a combination of volatile RAM for temporary storage and

processing, and non-volatile, programming read-only memory ("PROM") that contains permanent aspects of the controller's operating instructions.

The controller 14 also includes a series of input/output modules shown representatively at 20_1, 20_2 that sense the condition of, and send control signals to, the controlled machine (not shown) over a communication link (indicated by arrows). This communication link facilitates the bidirectional exchange of signals between each I/O module 20 and an associated device (e.g., a sensor or an actuator).

The controller 14 includes an internal bus provided by a backplane 22. The internal bus provides for operably connecting the modules of the programable logic controller 14 to each other via a local-area network. The communication of the modules over the internal bus can be accomplished by using a specific protocol such as, for example, the MODBUS Application Protocol by Schneider Automation, Inc.

The programable logic controller 14, and in particular the CPU module 19, provides for operating the I/O modules 20. The programable logic controller 14, via the I/O modules 20, examines the condition of selected sensing devices associated with the controlled equipment, and, based thereon, sends appropriate operative control signals to the equipment.

The instructions (i.e., symbols) for operation of the controller 14 are written in a relatively high-level language that can be stored in block 18 for permitting not only manipulation of input and output data, but also arithmetic and timing functions, file-handling capabilities and other complex tasks. These instructions are translated into machine-readable code by the controller via an interpreter 25. For example, one standardized type of instruction symbolically represents control functions according to a relay ladder diagram; it may be desired, however, to utilize state-control languages that represent controller actions in terms of steps,

each of which consists of a command that creates actions and one or more instructions for leaving the step.

Server 12 is operably connected to the backplane 22 of the programable logic controller 14 via a backplane driver 24. The server 12 allows Internet/Intranet access to information stored in the programable logic controller 12 including controller data, I/O diagnostics, configuration information through a "thin client" (i.e., web browsers). The server 12 is substantially "plug-and-play" wherein most of the server system is transparent to the user (web browser) and does not have a user interface in the traditional sense.

In particular, the server 12 allows Internet/Intranet access to controller and network information by having a HTTP sever 36 and a File Transfer Protocol (FTP) server 38 within the module 12. These servers provide a set of HTML pages that allow the user to display controller configuration, controller status, I/O module configuration, I/O module status, to view and modify register values and display the Ethernet statistics. The user is also allowed to customize Web pages and download them into the server.

The use of the web along with allowing for uploading customizable HMTL pages provides a low-cost ethernet connection Web Browser view to programable logic controller data and diagnostics.

As stated previously, the MODBUS protocol is preferably used to communicate between the server 12 and the controller 14. In addition, the server 12 communicates with the host 16 over an Ethernet network 46. Accordingly, the server 12 provides both a MODBUS on Ethernet Server 26 and a MODBUS on Ethernet Client 27 for providing communications between the controller 14 and the host 16.

The server 12 also includes a flash memory 28 operably connected via a flash memory driver 32 to both the controller backplane driver 24 and a file system

manager 33. Stored within the flash memory 28 are variable/symbol information that can be downloaded to the programable logic controller 14 for execution. The variable/symbol information is loaded into the memory via the FTP server 38 from a database or the like. The use of the FTP server 38 provides for the efficient transfer of variable/symbol information along with other program files into the memory 28.

The flash memory 28 also provides storage for supporting the HTTP server 36, the FTP server 38, and operation of TCP/IP. Moreover, the flash memory 28 stores JAVA applets and data structures defining one or more web pages shown representatively at 34_1, 34_2. The web pages 34 consists of ASCII data obtained from one or more of I/O modules 20, HTML formatting instructions and associated data, and/or "applet" instructions (i.e., JAVA code) for causing a properly equipped remote computer to display the data in a dynamic fashion.

In particular, to implement reading information from the controller 14 and displaying it in HTML pages, much of this information handling is done by the JAVA applets running on the browser of the host 16. Preferably, the applets make data requests of the controller 14 with ModBus commands over TCP/IP.

Management and transmission of web pages 40 to a querying computer is handled by the File System Manager 33 an the HTTP server 36 for allowing module 12 to function as a network server. The host or client 16 connects to the server 12 using the IP address of the HTTP server 36 assigned by the network administrator. The HTTP server 36 downloads HTML documents to the user's web browser from the flash memory 28 such as, for example, the "home page" of the server 12. If desired, the server 12, via the "home page," can prompt the user for a predefined password to allow access to additional web pages with "read-only" permission and another password to allow "read-write" permission. Thus, if

the user is knowledgeable in HTML, the appearance of the home page can be customized by downloading another "home page" into the flash memory 28.

As such, HTTP is the primary protocol used to communicate between the host computer 16 and the server 12. The HTTP server 36, commonly called a web server, is the "active component" of the server 12 that listens on a designated TCP/IP port for HTTP requests from web browsers. When a request for a Web page (HTML document) is sent to the HTTP server 36, the server retrieves, or dynamically creates, the appropriate page from the flash memory 28, via the file system manager 30, and transmits it to the browser using HTTP.

As previously indicated, FTP is another protocol used to communicate between a client and the server 12 which is handled primarily by block 38. The FTP server 38 is the "active component" of the module 12 that listens on a designated TCP/IP port for requests for programs stored within the flash memory 28. Furthermore, the FTP server 38 provides for uploading programs and web pages into the flash memory 28.

Incoming data from I/O modules 20 can be, if desired, processed by the programable logic controller 14 before being copied to one of the web pages 34 within the server 12. Because of the linking capabilities of the web, it is not necessary for the data to be stored in the web page containing the display instructions; instead, the latter page may contain a "hyperlink" pointer to a different web page in which data is accumulated. In addition, a web page can obtain data from other web pages (e.g., from different controllers) by accessing those web pages when appropriate. For example, if a cluster of controllers are operationally related such that data from one is usefully combined with data from the others, each page of the cluster can contain instruction to access the other pages (or their associated data pages) when accessed by a user, and the applet configured to present data from the entire cluster. Alternatively, the applet can be configured

to cause the client's browser to access the web page. As used herein, data is "associated with" a web page or an applet if it is stored as part of the web page or applet, or stored in a directly or indirectly hyperlinked web page.

Network communication blocks 40 and 41 provide for operating and connecting the sever module 12 to a host or client computer 16 over a local-area network. Accordingly, communication blocks 40 and 41 include data-transmission circuitry to transfer streams of digitally encoded data over telephone or other communication lines.

In an embodiment, computer 16 can function as a network client and consists of a Personal Computer (PC) running a WINDOWS based graphical user interface supplied by Microsoft Corporation. Computer 16 also includes a network interface for facilitating connection to and data transfer through the computer network 46 which can be a local network, the Internet, or an Internet-linked local network. Naturally, computer 16 also contains various conventional components, i.e., a display, a storage system, an operating system and a graphical user interface, and a keyboard and/or position-sensing device (e.g., a mouse) for accepting input from the user. For convenience of presentation, these are not shown.

Also loaded onto the computer 16 is a web browser 44 that supports JAVA, such as INTERNET EXPLORER (supplied by Microsoft Corp.) or NETSCAPE NAVIGATOR (supplied by America On Line) and communicates with the server 12, which in turn communicates with the programable logic controller 14. Preferably, the host computer 16 and the server 12 communicate with each other via an Ethernet TCP/IP network 46.

In particular, the web pages 34 within the flash memory 28 enable a user of the host computer 16 to view certain information about the controller 14 without a custom client application. For instance, if the host computer 16 includes a standard web browser 44 and access to the server 12, then a user of the host can

view the contents of the controller 14 connected to the server by downloading a page 30 stored within the flash memory 28.

In another example, the host computer 16, via the web browser 44, sends requests to the server 12 to view a set of controller registers within the controller CPU module. The server 12 then queries the controller 14 via the MODBUS on Ethernet Client for the requested information. The controller 14 responds to the query by sending the requested data to the sever 12. Finally, via the MODBUS on Ethernet Server 26, the sever 12 sends this information back to the web browser client 16 that requested it.

As indicated above, the web pages 34 within the flash memory 28 allow a user of the web browser 44 to "browse" the contents of a programable logic controller 12 as if it was just another web site. After the web page has been downloaded to the host 14, users can use their web browser to connect to the programable logic controller and view controller information such as its status, I/O health, configuration information, and runtime values of variables and registers.

Preferably, an Application Programming Interface (API) is provided for writing applets or applications, preferably in JAVA (Trademark of Sun Microsystems, Inc., Mountain View, CA), that communicate with the controller 14 via Ethernet TCP/IP. The application programming interface includes several classes for writing applets and applications and an archive containing all the class files needed to compile an applet or application written by a user. Preferably, the archive is not platform dependent so compiling applets for different types of controllers (i.e., controller executing various control programs) is supported.

A desirable feature of JAVA is that applet security restrictions permit an applet to connect only to the host from which is was downloaded. Accordingly, when developing an applet for downloading to a server or controller, it is necessary that the classes comprising the applet be put in an archive file and downloaded to the

server or controller. Moreover, an HTML file (i.e., web page) that contains an <APPLET>tag for the applet should also be downloaded. Accordingly, a web browser can than be used to load the web page from the server or controller and view the applet.

In an embodiment, as shown in FIGURE 2, the application programming interface 60 includes a communications interface 62 and a communications library 64 for allowing a developer, preferably using JAVA, to easily communicate with a controller using TCP/IP. The communications interface 62 is preferably a conventional JAVABEAN interface operably connected between the communication client 16 and the controller 14. Accordingly, all of the classes within the communications interface 62 are preferably implemented as JAVABEANS. The Beans can be connected by use of a 'Bean Box' or by writing Java code to connect them directly. The methods of the classes provided by the JAVABEAN interface 62 are preferably the only ones normally needed by a client programmer to use all of the functions available from the communications library 64.

The communication library 64 is operably connected between the communications interface 62 and the programable logic controller 14. The communications library 64 provides support for communicating with the programable logic controller 14 and accessing information about variables in a program being executed by the controller. This allows the communications library 64 to provide on-demand reading and writing of a variable's values, as well as continuous monitoring of its value and reporting to the client 16 any changes in the value.

The primary interface for the communications library 64 is a set of JavaBeans. These Beans can be generally categorized into a communication bean, a family of get and set beans, and a family of monitor beans. The communication bean is the

main Bean that an applet or application instantiate in order to establish a connection to the controller 14. The family of get and set Beans gets and sets the run-time values of variables, or register/memory addresses, of a controller's program 'on-demand.' Furthermore, the monitor family of Beans continuously monitors the run-time value of variables or register/memory addresses, of a controller's program and notifies the client when the values change.

The organization of the classes within the communications library 64 is shown in FIGURE 3. The communications library 64 includes a communications package 66 and a namespace package 68. The communications package 66 includes elements comprising a client handler 70, a server handler 72, a value adapter 74, a subscription list 76, and an update queue 78.

The client handler 70 is operably attached to the communication client 16 and maintains the interface with the client by receiving instructions and then, based on the instructions, providing appropriate responsive input signals to elements within the communications package 66 and the namespace package 68 .

The server handler 72 is operably attached to the value adaptor 74 and the update queue 78. The server handler 72 provides for unsolicited data updates by receiving data update notifications from the programable logic controller 14 and then forwarding this information to the value adapter 74.

The value adapter 74 is operably attached to the server handler 72, the client handler 70, and the communication client 16. The value adapter 74 dispatches messages for unsolicited data updates to the client 16. In particular, the value adapter 74 receives messages from the server handler 72 and then forwards the notification.

The subscription list 76 is operably attached to the update queue 78 and the client handler 70. The subscription list 76 handles the polling of the programable

logic controller 12 for those data items that the communication client 14 has requested event notification when the data item's values changes.

The update queue 78 is operably attached to the server handler 72, subscription list 76, and the client handler 70. The update queue 78 handles the dispatching of the event notifications that are sent by the subscription list 76. When an event notification is sent to the update queue 78, it is placed in a first-in-first-out queue. Accordingly, the update queue 78 provides for a buffering of the update events that are sent by the subscription list 76. In this way the rate at which the programable logic controller 14 can be polled is decoupled from the rate at which the display (not shown) can be updated on the client 16.

The namespace package 68 includes an element comprising a variable lookup adaptor 80 operably attached to the client 16, the client handler 70, and the programable logic controller 14. The variable lookup adaptor 80 executes instructions from the client handler 70 for obtaining information about variables contained in programs executed by controller 14.

Within FIGURE 3, elements within the communication package 66 and the namespace package 68 are indicated as residing on the client side, server side, or on both. The client side within FIGURE 3 indicates those elements that are executed by the user's web browser. Further, the server side indicates those elements that are executed by a server or, if a server is not used in communicating with the controller 14, the user's web browser.

In an embodiment, the interface includes several packages that are indexed, for example, to include: com.package.dt; com.package.main; com.package.main.comm; com.package.namespace; and com.package.vars.

Package com.package.dt includes classes com.package.dt.DT and com.package.dt.FT. Preferably, class com.package.dt.DT has a public class DT

and extends Object. In particular, the DT class defines the platform-independent data types for PLC references. The variables for class com.package.dt.DT include: BCD16 - Unsupported data type (provided for mapping of different PLC data types if needed); BCD32 - Unsupported data type (provided for mapping of different PLC data types if needed); BCD64 - Unsupported data type (provided for mapping of different PLC data types if needed); BCD8 - Unsupported data type (provided for mapping of different PLC data types if needed); BOOL - DT value for the boolean data type; DATE - Unsupported data type (provided for mapping of different PLC data types if needed); DINT - DT value for the double integer data type; DT- Unsupported data type (provided solely for mapping of different PLC data types if needed); INT - DT value for the integer data type; LONG - Unsupported data type (provided for mapping of different PLC data types if needed); LREAL - Unsupported data type (provided for mapping of different PLC data types if needed); REAL - DT value for the floating point data type; SHORT - DT value for the short data type; STR - DT value for the string data type; TIME - DT value for the time data type; TOD - Unsupported data type (provided for mapping of different PLC data types if needed); typeNames - Array of names for the supported data types; UDINT - DT value for the unsigned double integer data type; UINT - DT value for the 'unsigned integer data type; ULONG - Unsupported data type (provided for mapping of different PLC data types if needed); UNDEFINED - DT value for the 'undefined/unknown' data type; and USHORT - DT value for the unsigned short data type.

In this embodiment, the constructor for class com.package.dt.DT is "DT()" and the methods are: getSize(short) for returning the size (in bytes) of a specified data type; toDT(String) for returning the data type (DT value) for a specified data type name; and toName(short) for returning the identifying text string (name) for a specified data type.

Class com.package.dt.FT has a public class FT and extends Object. Preferably, the variables for class com.package.dt.FT include: ASCII, BIN, BOOL, DEC, HEX, N_FMTS, NOFMT, REAL, and TIME. In this embodiment, the constructor for class com.package.dt.FT is "FT()" and the methods are: formatValue(Number, int, short) and parseValue(String, int, short).

Package com.package.main.comm includes the class com.package.main.comm.StatusMessages having a public class StatusMessages and extending Object. In this embodiment, the constructor for class com.package.main.comm is "StatusMessages()" and the methods are: get(int), get(Number), and init(Locale).

Package com.package.main includes classes: com.package.main.CommBean; com.package.main.CommBeanVarLookup; com.package.main.ConnectPLC; com.package.main.GetBits; com.package.main.GetBool; com.package.main.GetDInt; com.package.main.GetInt; com.package.main.GetNumber; com.package.main.GetReal; com.package.main.GetRef; com.package.main.GetShort; com.package.main.GetString; com.package.main.GetUDInt; com.package.main.GetUInt; com.package.main.GetUShort; com.package.main.LiveLabelApplet; com.package.main.LiveLabelMgrApplet; com.package.main.MonitorAdapter; com.package.main.MonitorBits; com.package.main.MonitorBool; com.package.main.MonitorDInt; com.package.main.MonitorInt; com.package.main.MonitorNumber; com.package.main.MonitorReal; com.package.main.MonitorRef; com.package.main.MonitorShort; com.package.main.MonitorString; com.package.main.MonitorUDInt; com.package.main.MonitorUInt; com.package.main.MonitorUShort; com.package.main.ReadRef; com.package.main.Ref; com.package.main.ServerChangeEvent;

com.package.main.ServerChangeListener; com.package.main.SetBits; com.package.main.SetBool; com.package.main.SetDInt; com.package.main.SetInt; com.package.main.SetNumber; com.package.main.SetReal; com.package.main.SetRef; com.package.main.SetShort; com.package.main.SetString; com.package.main.SetUDInt; com.package.main.SetUInt; and com.package.main.SetUShort.

Class com.package.main.CommBean has a public class CommBean, extends Object, and implements Serializable. In particular, this class preferably is the support class for all Main Beans. Every Main Bean must be provided a reference to this class either by becoming a ServerChangeListener (if using a Bean Box) or via the Bean's constructor (if using Java code to connect the Beans).

In this embodiment, the constructor for class com.package.main.CommBean is "CommBean()", as a default constructor, and "CommBean(Locale)" as a constructor to set a locale for status and exception messages. Moreover, the methods are: addPropertyChangeListener(PropertyChangeListener) for adding a property change listener; addServerChangeListener(ServerChangeListener) for adding a 'server change' listener; connect(String, boolean) for creating a connection to a PLC; disconnect() for disconnecting the connection with a PLC; getAdaptor() for returning reference to the current 'value adapter'; getServer() for returning reference to the current 'comm server'; getVarLookup() for returning reference to the current 'namespace server'; isConnected() for determining if connection to PLC has been established; isStarted() for determining if registered data items have been subscribed; isSuspended() for determining if processing of subscription list has been suspended; removePropertyChangeListener(PropertyChangeListener) for removing a property change listener; removeServerChangeListener(ServerChangeListener) for removes a 'server change' listener; resume() for resuming processing of the subscription list; start()

for subscribing (starting) all registered data items; stop() for unsubscribing (stoping) all registered data items; and suspend() for suspending processing of the subscription list.

Class com.package.main.CommBeanVarLookup has a public class of "CommBeanVarLookup", extends Object, and implements VarLookupInterface. In particular, the CommBeanVarLookup class is used to access the variables of PLC databases that have been loaded from a Symbol Table (namespace) file stored on the PLC.

In this embodiment, the constructor for class com.package.main.CommBeanVarLookup is "CommBeanVarLookup()", as a default constructor, and "CommBeanVarLookup(Locale)" as a constructor to set a locale. Moreover, the methods are: get(String) for returning the attributes of a specified variable (symbol); getSymbolCount() for returning the number of symbols (variables) that were loaded from the Symbol Table (name space); getSymbols() for returning the list of symbols (variables) that were loaded from the Symbol Table (name space); getVerInfo() for returning the version of the PLC database that was used to create the Symbol Table (namespace) file that is in the connected PLC; init(InetAddress) for causing the Symbol Table (name space) to be loaded from the specified host; isReadOnly(String, int) for determining if register reference (direct address) has been designated as read-only.

Class com.package.main.ConnectPLC has a public class of "ConnectPLC", extends Object, and implements ServerChangeListener, PropertyChangeListener, Serializable. In particular, the ConnectPLC class is a Bean that is used to specify the PLC to connect to. This class is intended for use in a Bean Box only. (For connecting to a PLC via direct Java code, use the connect() method of class CommBean.)

In this embodiment, the constructor for class com.package.main.ConnectPLC is "ConnectPLC" and the methods are: addPropertyChangeListener(PropertyChangeListener) for adding a property change listener; connect() for creating connection to the PLC; disconnect() for disconnecting the connection with the PLC; getPLC() for returning name of PLC that is to be connected; isConnected() for determines if connection to PLC has been established; isLoadSymbols() for determining if Symbol Table (namespace) is to loaded upon connection; isStarted() for determining if registered data items have been subscribed; isSuspended() for determines if processing of subscription list has been suspended; propertyChange(PropertyChangeEvent) which is invoked when a bound property (of CommBean) is changed; removePropertyChangeListener(PropertyChangeListener) for removes a property change listener; resume() for resuming processing of the subscription list; serverChanged(ServerChangeEvent) which is invoked when the comm server changes; setLoadSymbols(boolean) for seting flag that will cause Symbol Table to be loaded; setPLC(String) for setting name of PLC that is to be connected; start() for subscribing (starting) all registered data items; stop() for unsubscribing (stoping) all registered data items; suspend() for suspending processing of the subscription list.

Class com.package.main.GetBits has a public class of "GetBits", extends GetRef, and implements Serializable. In particular, the GetBits class is a Bean that is used to read a variable or register(s) reference 'on demand'. The retrieved value is available as a Java BitSet. The size of the resultant BitSet is determined by multiplying the quantity of references times the size (in bits) of each reference. The size of each allowed data type is: BOOL 1 bit SHORT 8 bits USHORT 8 bits INT 16 bits UINT 16 bits DINT 32 bits UDINT 32 bits If the address specified for the data item is a variable name, the quantity of references is one (the default).

Also, for all variable references, it is not necessary to set the data type of the reference. (A variable's data type is looked-up as part of processing the read request.) If, however, the data type of a variable reference is explicitly set, then it matches exactly the actual data type of the variable.

In this embodiment, the constructor for class com.package.main.GetBits is "GetBits", as the default constructor, and "GetBits(ClientHandlerInterface) as the constructor to set the 'comm server' to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValueAsString() for returning the retrieved value; and setDatatype(short) for setting the data type of the reference that is to be read from PLC.

Class com.package.main.GetBool has a public class of "GetBool", extends GetRef, and implements Serializable. In particular, the GetBool class is a Bean that is used to read a variable or discrete(s) reference 'on demand'. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is BOOL.

In this embodiment, the constructor for class com.package.main.GetBool is "GetBool()", as the default constructor, and "GetBool(ClientHandlerInterface) as the constructor to set the 'comm server' to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array of retrieved values.

Class com.package.main.GetDInt has a public class of "GetDInt", extends GetRef, and implements Serializable. In particular, the GetDInt class is a Bean that is used to read a variable or register(s) reference 'on demand'. Each reference is treated as a signed, 32-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is DINT.

In this embodiment, the constructor for class com.package.main.GetDInt is "GetDInt()", as the default constructor, and "GetDInt(ClientHandlerInterface)" as the constructor to set the 'comm server' to be used for processing requests. In addition, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array of retrieved values.

Class com.package.main.GetInt has a public class of "GetInt", extends GetRef, and implements Serializable. In particular, the GetInt class is a Bean that is used to read a variable or register(s) reference 'on demand'. Each reference is treated as a signed, 16-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is INT.

In this embodiment, the constructor for class com.package.main.GetInt is "GetInt()", as the default constructor, and "GetInt(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array of retrieved values.

Class com.package.main.GetNumber has a public class of "GetNumber", extends GetRef, and implements Serializable. In particular, the GetNumber class is a Bean that is used to read a variable or register(s) reference 'on demand'. The data type of the reference(s) is a settable property. If the address specified for the data item is a variable name, the quantity of references is one (the default). Also, for all variable references, it is not necessary to set the data type of the reference. (A variable's data type is looked-up as part of processing the read request.) If, however, the data type of a variable reference is explicitly set, then it should match the actual data type of the variable.

In this embodiment, the constructor for class com.package.main.GetNumber is "GetNumber()", as the default constructor, and "GetNumber(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); getValues(int) for returning a specific element of the array of retrieved values; and getDatatype(short) for setting the data type of the reference that is to be read from the PLC.

Class com.package.main.GetReal has a public class of "GetReal", extends GetRef, and implements Serializable. In particular, the GetReal class is a Bean that is used to read a variable or register(s) reference 'on demand'. Each reference is treated as a 32-bit IEEE floating point number. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is REAL.

In this embodiment, the constructor for class com.package.main.GetReal is "GetReal()", as the default constructor, and "GetReal(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array of retrieved values.

Class com.package.main.GetRef has a public class of "GetRef", extends ReadRef, and implements ServerChangeListener and Serializable. In particular, the GetRef class is the common base class for all the 'Get' Beans. The methods of this class are: doAction() for initiating the reading of values from the PLC; readValues() for initiating the reading of values from the PLC; and serverChanged(ServerChangeEvent) for invoking when the comm server changes.

Class com.package.main.GetShort has a public class of "GetShort", extends GetRef, and implements Serializable. In particular, the GetShort class is a Bean that is used to read a variable or register(s) reference 'on demand'. Each reference is treated as a signed, 8-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is SHORT.

In this embodiment, the constructor for class com.package.main.GetShort is "GetShort()", as the default constructor, and "GetShort(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array of retrieved values.

Class com.package.main.GetString has a public class of "GetString", extends GetRef, and implements Serializable. In particular, the GetString class is a Bean that is used to read the bytes in a set of registers 'on demand' and interpret them as an 8-bit ASCII character string. The quantity of references to be read is set to the number of characters to be read, not the number of registers.

In this embodiment, the constructor for class com.package.main.GetString is "GetString()", as the default constructor, and "GetString(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred method is getValue() for returning the retrieved value.

Class com.package.main.GetUDInt has a public class of "GetUDInt", extends GetRef, and implements Serializable. In particular, the GetUDInt class is a Bean that is used to read a variable or register(s) reference 'on demand'. Each reference is treated as an unsigned, 32-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is UDINT.

In this embodiment, the constructor for class com.package.main.GetUDInt is "GetUDInt()", as the default constructor, and "GetUDInt(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; and getValues() for returning the retrieved value(s).

Class com.package.main.GetUInt has a public class of "GetUInt", extends GetRef, and implements Serializable. In particular, the GetUInt class is a Bean that is used to read a variable or register(s) reference 'on demand'. Each reference is treated as an unsigned, 16-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is UINT.

In this embodiment, the constructor for class com.package.main.GetUInt is "GetUInt()", as the default constructor, and "GetUInt(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array or retrieved values.

Class com.package.main.GetUShort has a public class of "GetUShort", extends GetRef, and implements Serializable. In particular, the GetUShort class is a Bean that is used to read a variable or register(s) reference 'on demand'. Each reference is treated as an unsigned, 8-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is USHORT.

In this embodiment, the constructor for class com.package.main.GetUShort is "GetUShort()", as the default constructor, and "GetUShort(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are:

getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array or retrieved values.

Class com.package.main.LiveLabelApplet has a public class of "LiveLabelApplet", extends Applet, and implements PropertyChangeListener. In this embodiment, the constructor for class com.package.main.LiveLabelApplet is "LiveLabelApplet()" and the preferred methods are: init(), propertyChange(PropertyChangeEvent), and start().

Class com.package.main.LiveLabelMgrApplet has a public class of "LiveLabelMgrApplet", extends Applet, and implements Runnable. In this embodiment, the constructor for class com.package.main.LiveLabelMgrApplet is "LiveLabelMgrApplet()" and the preferred methods are: destory(), getCommBean(), init(), run(), start(), and stop().

Class com.package.main.MonitorAdapter has a public class of "MonitorAdapter", extends Object, and implements PropertyChangeListener and Serializable. In particular, the Monitor class, when connected as a PropertyChangerListener of a "Monitor Bean," converts the Bean's dynamic properties to Strings.

In this embodiment, the constructor for class com.package.main.MonitorAdapter is "MonitorAdapter()" and the preferred methods are: addPropertyChangeListener(PropertyChangeListener) for adding a property change listener; getRegistered() for determining if the reference is registered for continuously monitoring; getStatus() for returning the current 'acquisition status' message for the reference; getSubscribed() for determining if the reference is currently being continuously monitored; getValue() for returning the numeric value retrieved from PLC as a string (decimal representation); getValues() for providing access to the numeric values retrieved from PLC as an

array of strings (decimal representation); getValues(int) for providing access to a specific element of an array of retrieved values; propertyChange(PropertyChangeEvent) which is invoked when a bound property (of the associated "comm bean") is changed; and removePropertyChangeListener(PropertyChangeListener) for removing a property change listener.

Class com.package.main.MonitorBits has a public class of "MonitorBits", extends MonitorRef, and implements Serializable. In particular, the MonitorBits class is a Bean that is used to read continuously a variable or register(s) reference. The size of the resultant BitSet is determined by multiplying the quantity of references times the size (in bits) of each reference. The size of each allowed data type is: BOOL 1 bit SHORT 8 bits USHORT 8 bits INT 16 bits UINT 16 bits DINT 32 bits UDINT 32 bits. If the address specified for the data item is a variable name, the quantity of references is one (the default). Also, for all variable references, it is not necessary to set the data type of the reference. (A variable's data type is always looked-up as part of processing the read request.) If, however, the data type of a variable reference is explicitly set, then is should match the actual data type of the variable.

In this embodiment, the constructor for class com.package.main.MonitorBits is "MonitorBits()", as the default constructor, and "MonitorBits(ValueAdaptor)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValuesAsStrings() for returning the retrieved value; and setDatatype(short) for setting the data type of the reference that is to be read from PLC.

Class com.package.main.MonitorBool has a public class of "MonitorBool", extends MonitorRef, and implements Serializable. In particular, the MonitorBool class is a Bean that is used to read continuously a variable or discrete(s) reference.

If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is BOOL.

In this embodiment, the constructor for class com.package.main.MonitorBool is "MonitorBool()", as the default constructor, and "MonitorBool(ValueAdaptor)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array or retrieved values.

Class com.package.main.MonitorDInt has a public class of "MonitorDInt", extends MonitorRef, and implements Serializable. In particular, the MonitorDInt class is a Bean that is used to read continuously a variable or register(s) reference. Each reference is treated as a signed, 32-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is DINT.

In this embodiment, the constructor for class com.package.main.MonitorDInt is "MonitorDInt()", as the default constructor, and "MonitorDInt(ValueAdaptor)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array of retrieved values.

Class com.package.main.MonitorInt has a public class of "MonitorInt", extends MonitorRef, and implements Serializable. In particular, the MonitorInt class is a Bean that is used to read continuously a variable or register(s) reference. Each reference is treated as a signed, 16-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is INT.

In this embodiment, the constructor for class com.package.main.MonitorInt() is "MonitorInt()", as the default constructor, and "MonitorInt(ValueAdaptor)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array of retrieved values.

Class com.package.main.MonitorNumber has a public class of "MonitorNumber", extends MonitorRef, and implements Serializable. In particular, the MonitorNumber class is a Bean that is used to read continuously a variable or register(s) reference. The data type of the reference(s) is a settable property. If the address specified for the data item is a variable name, the quantity of references is one (the default). Also, for all variable references, it is not necessary to set the data type of the reference. (A variable's data type is looked-up as part of processing the read request.) If, however, the data type of a variable reference is explicitly set, then is should match the actual data type of the variable.

In this embodiment, the constructor for class com.package.main.MonitorNumber is "MonitorNumber()", as the default constructor, and "MonitorNumber(ValueAdaptor)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); getValues(int) for returning a specific element of the array of retrieved values; and setDatatype(short) for setting the data type of the reference that is to be read from PLC.

Class com.package.main.MonitorReal has a public class of "MonitorReal", extends MonitorRef, and implements Serializable. In particular, the MonitorReal class is a Bean that is used to read continuously a variable or register(s) reference. Each reference is treated as a 32-bit IEEE floating point number. If the address

specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is REAL.

In this embodiment, the constructor for class com.package.main.MonitorReal is "MontorReal()", as the default constructor, and "GetReal(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array of retrieved values.

Class com.package.main.MonitorRef has a public class of "MonitorRef", extends ReadRef, and implements ServerChangeListener and Serializable. In particular, the MonitorRef class is the common base for all the 'Monitor' Beans. Moreover, the preferred methods of this class are: deregister() for deregistering the data item for continuous monitoring; getSubscribeStatus() for returning the 'Subscribe Status' for the reference; getXactID() for returning the 'Transaction ID' for the reference; isRegistered() for determining if the reference is registered for continuously monitoring; isSubscribed() for determining if the reference is currently being continuously monitored; register() for registering the data item for continuous monitoring; serverChanged(ServerChangeEvent) which is invoked when the comm server changes; setAddress(String) for setting the address for the reference; setQuantity(short) for setting the number of data items to be monitored; updateValue(ValueChangeEvent) which is invoked by 'comm server' when the value or status of the monitored data item changes.

Class com.package.main.MonitorShort has a public class of "MonitorShort", extends MonitorRef, and implements Serializable. In particular, the MonitorShort class is a Bean that is used to read continuously a variable or register(s) reference. Each reference is treated as a signed, 8-bit integer. If the address specified for the

data item is a variable name, the quantity of references is one (the default), and the variable's data type is SHORT.

In this embodiment, the constructor for class com.package.main.MonitorShort is "MonitorShort()", as the default constructor, and "MonitorShort(ValueAdaptor)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array or retrieved values.

Class com.package.main.MonitorString has a public class of "MonitorString", extends MonitorRef, and implements Serializable. In particular, the MonitorString class is a Bean that is used to read continuously the bytes in a set of registers and interpret them as an 8-bit ASCII character string. The quantity of references to be read is set to the number of characters to be read, not the number of registers.

In this embodiment, the constructor for class com.package.main.MonitorString is "MonitorString()", as the default constructor, and "MonitorString(ValueAdaptor)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred method is getValue() for returning the retrieved value.

Class com.package.main.MonitorUDInt has a public class of "MonitorUDInt", extends MonitorRef, and implements Serializable. In particular, the MonitorUDInt class is a Bean that is used to read continuously a variable or register(s) reference. Each reference is treated as an unsigned, 32-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is UDINT.

In this embodiment, the constructor for class com.package.main.MonitorUDInt is "MonitorUDInt()", as the default constructor, and "MonitorUDInt(ValueAdaptor)" as the constructor to set the "comm server" to be

used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; and getValues() for returning the retrieved value(s).

Class com.package.main.MonitorUInt has a public class of "MonitorUInt", extends MonitorRef, and implements Serializable. In particular, the MonitorUInt class is a Bean that is used to read continuously a variable or register(s) reference. Each reference is treated as an unsigned, 16-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is UINT.

In this embodiment, the constructor for class com.package.main.MonitorUInt is "MonitorUInt()", as the default constructor, and "MonitorUInt(ValueAdaptor)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; and getValues() for returning the retrieved value(s).

Class com.package.main.MonitorUShort has a public class of "MonitorUShort", extends MonitorGetRef, and implements Serializable. In particular, the MonitorUShort class is a Bean that is used to read continuously a variable or register(s) reference. Each reference is treated as an unsigned, 8-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is USHORT.

In this embodiment, the constructor for class com.package.main.MonitorUShort is "MonitorUShort()", as the default constructor, and "MonitorUShort(ValueAdaptor)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the retrieved value; getValues() for returning the retrieved value(s); and getValues(int) for returning a specific element of the array or retrieved values.

Class com.package.main.ReadRef has a public class of "ReadRef", extends Ref, and implements Serializable. In particular, the ReadRef class is the common base for all the 'Get/Monitor' Beans. The preferred methods for this class are: getQuantity() for returning the number of references that are to be read from the PLC; getValueAsString() for returning the retrieved value; and setQuantity(short) for setting the number of references to be read from the PLC.

Class com.package.main.Ref has a public abstract class of "Ref", extends Object, and implements Serializable. In particular, the Ref class is the common base class for all the 'Get/Monitor' Beans. The preferred methods for this class are: addPropertyChangeListener(PropertyChangeListener) for adding a property change listener; getAddress() for returning the address that has been set for the reference; getDatatype() for returning the data type that has been set for the reference; getStatus() for returns the current 'acquisition status' code for the reference; removePropertyChangeListener(PropertyChangeListener) for removing a property change listener; and setAddress(String) for setting the address for the reference.

Class com.package.main.ServerChangeEvent has a public class of "ServerChangeEvent", extends EventObject, and implements Serializable. In particular, this class is the event object that is fired when the 'comm server' is changed. This class is preferably needed only for connecting the individual Get/Set/Monitor Beans with an instance of CommBean in a Bean Box.

In this embodiment, the constructor for class com.package.main.SeverChangeEvent is "ServerChangeEvent(Object, ClientHandlerInterface, ValueAdaptor, VarLookupInterface)". Moreover, the preferred methods are: getAdaptor() for returning reference to the current 'value adapter'; getServer() for returning reference to the current 'comm server'; and getVarLookup() for returning reference to the current 'namespace server'.

Class com.package.main.ServerChangeListener has a public interface of "ServerChangeListener" and extends EventListener. In particular, this interface identifies a class capable of receiving ServerChangeEvent objects. Preferably, this interface is used to 'connect' an instance of CommBean to the individual Get/Set/Monitor Beans, which all implement this interface. This interface is preferably needed only for connecting Beans in a Bean Box. The preferred method is serverChanged(ServerChangeEvent) which is invoked when the 'comm server' changes.

Class com.package.main.SetBits has a public class of "SetBits", extends SetRef, and implements Serializable. In particular, the SetBits class is a Bean that is used to write a variable or register(s) reference 'on demand'. The value to be written is specified using a Java BitSet. The size of the BitSet is determined by multiplying the quantity of references times the size (in bits) of each reference. The size of each allowed data type is: BOOL 1 bit SHORT 8 bits USHORT 8 bits INT 16 bits UINT 16 bits DINT 32 bits UDINT 32 bits If the address specified for the data item is a variable name, the quantity of references is one (the default). Also, for all variable references, it is not necessary to set the data type of the reference. (A variable's data type is looked-up as part of processing the read request.) If, however, the data type of a variable reference is explicitly set, then it matches exactly the actual data type of the variable.

In this embodiment, the constructor for class com.package.main.SetBits is "SetBits", as the default constructor, and "SetBits(ClientHandlerInterface) as the constructor to set the 'comm server' to be used for processing requests. Moreover, the preferred methods are: getQuantity() for returning the number of references to be written to the PLC; getValue() for returning the value to be written to the PLC; getValueAsString() for returning the value to be written to the PLC; sendValue() for initiating the sending of the value to the PLC; setDatatype(short) for setting the

data type of the reference that is to be written to the PLC; setQuantity(short) for setting the number of references (NOT the number of bits) to be written to the PLC; and setValue(BitSet) for setting the value to be written to the PLC.

Class com.package.main.SetBool has a public class of "SetBool", extends SetRef, and implements Serializable. In particular, the SetBool class is a Bean that is used to write a variable or discrete(s) reference 'on demand'. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is BOOL.

In this embodiment, the constructor for class com.package.main.SetBool is "SetBool()", as the default constructor, and "SetBool(ClientHandlerInterface) as the constructor to set the 'comm server' to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the value to be written to the PLC; getValues() for returning the values to be written to the PLC; getValues(int) for returning a specific element of the array of values to be written to the PLC; getValue(boolean) for setting a single value to be written to the PLC; getValues(boolean[]) for setting the values to be written to the PLC; and setValues(int,boolean) for setting a specific element of the array of values to be written to the PLC.

Class com.package.main.SetDInt has a public class of "SetDInt", extends SetRef, and implements Serializable. In particular, the SetDInt class is a Bean that is used to write a variable or register(s) reference 'on demand'. Each reference is treated as a signed, 32-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is DINT.

In this embodiment, the constructor for class com.package.main.SetDInt is "SetDInt()", as the default constructor, and "SetDInt(ClientHandlerInterface)" as the constructor to set the 'comm server' to be used for processing requests. In

addition, the preferred methods are: getValue() for returning the value to be written to the PLC; getValues() for returning the values to be written to the PLC; getValues(int) for returning a specific element of the array of values to be written to the PLC; getValue(int) for setting a single value to be written to the PLC; getValues(int, int) for setting a specific element of the array of values to be written to the PLC; and getValues(int[]) for setting the values to be written to the PLC

Class com.package.main.SetInt has a public class of "SetInt", extends SetRef, and implements Serializable. In particular, the SetInt class is a Bean that is used to write a variable or register(s) reference 'on demand'. Each reference is treated as a signed, 16-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is INT.

In this embodiment, the constructor for class com.package.main.SetInt is "SetInt()", as the default constructor, and "SetInt(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the value to be written to the PLC; getValues() for returning the values to be written to the PLC; getValues(int) for returning a specific element of the array of values to be written to the PLC; getValue(short) for setting a single value to be written to the PLC; getValues(int, short) for setting a specific element of the array of values to be written to the PLC; and getValues(short[]) for setting the value to be written to the PLC.

Class com.package.main.SetNumber has a public class of "SetNumber", extends SetRef, and implements Serializable. In particular, the SetNumber class is a Bean that is used to write a variable or register(s) reference 'on demand'. The data type of the reference(s) is a settable property. If the address specified for the data item is a variable name, the quantity of references is one (the default). Also, for all variable references, it is not necessary to set the data type of the reference.

(A variable's data type is looked-up as part of processing the read request.) If, however, the data type of a variable reference is explicitly set, then it should match the actual data type of the variable.

In this embodiment, the constructor for class com.package.main.SetNumber is "SetNumber()", as the default constructor, and "SetNumber(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the value to be written to the PLC; getValues() for returning the values to be written to the PLC; getValues(int) for returning a specific element of the array of values to be written to the PLC; setDatatype(short) for setting the data type of the reference that is to be written to the PLC; getValue(Number) for setting a single value to be written tot he PLC; getValues(int, Number) for setting a specific element of the array of values to be written to the PLC; and getValues(Number(]) for setting the values to be written to the PLC.

Class com.package.main.SetReal has a public class of "SetReal", extends SetRef, and implements Serializable. In particular, the SetReal class is a Bean that is used to write a variable or register(s) reference 'on demand'. Each reference is treated as a 32-bit IEEE floating point number. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is REAL.

In this embodiment, the constructor for class com.package.main.SetReal is "SetReal()", as the default constructor, and "SetReal(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the value to be written to the PLC; getValues() for returning the values to be written to the PLC; getValues(int) for returning a specific element of the array of values to be written

to the PLC; getValue(float) for setting a single value to be written to the PLC; getValues(float[]) for setting the values to be written to the PLC; and getValues(int, float) for setting a specific element of the array of values to be written to the PLC

Class com.package.main.SetRef has a public class of "SetRef", extends Ref, and implements ServerChangeListener and Serializable. In particular, the SetRef class is the common base class for all the 'Set' Beans. The methods of this class are: doAction() for initiating the sending of value(s) to the PLC; getQuantity for returning the number of references to be written to the PLC; sendValues() for initiating the sending of the value(s) to the PLC; and serverChanged(ServerChangeEvent) for invoking when the comm server changes.

Class com.package.main.SetShort has a public class of "SetShort", extends SetRef, and implements Serializable. In particular, the SetShort class is a Bean that is used to write a variable or register(s) reference 'on demand'. Each reference is treated as a signed, 8-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is SHORT.

In this embodiment, the constructor for class com.package.main.SetShort is "SetShort()", as the default constructor, and "SetShort(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the value to be written to the PLC; getValues() for returning the values to be written to the PLC; getValues(int) for returning a specific element of the array of values to be written to the PLC; getValue(byte) for setting a single value to be written to the PLC; getValues(byte[]) for setting the values to be written to the PLC; and getValues(int, byte) for setting a specific element of the array of values to be written to the PLC.

Class com.package.main.SetString has a public class of "SetString", extends SetRef, and implements Serializable. In particular, the SetString class is a Bean that is used to write a String to a set of registers 'on demand'. The String is written as a series of 8-bit ASCII characters. If the address specified for the data item is a variable name, the variable's data type is STR.

In this embodiment, the constructor for class com.package.main.SetString is "SetString()", as the default constructor, and "SetString(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the value to be written to the PLC; and setValue(String) for setting the value to be written to the PLC.

Class com.package.main.SetUDInt has a public class of "SetUDInt", extends SetRef, and implements Serializable. In particular, the SetUDInt class is a Bean that is used to write a variable or register(s) reference 'on demand'. Each reference is treated as an unsigned, 32-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is UDINT.

In this embodiment, the constructor for class com.package.main.SetUDInt is "SetUDInt()", as the default constructor, and "SetUDInt(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the value to be written to the PLC; getValues() for returning the values to be written to the PLC; getValues(int) for returning a specific element of the array of values to be written to the PLC; getValue(long) for setting a single value to be written to the PLC; setValues(int, long) for setting a specific element of the array of values to be written to the PLC; and getValues(long[]) for setting the values to be written to the PLC.

Class com.package.main.SetUInt has a public class of "SetUInt", extends SetRef, and implements Serializable. In particular, the SetUInt class is a Bean that is used to write a variable or register(s) reference 'on demand'. Each reference is treated as an unsigned, 16-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is UINT.

In this embodiment, the constructor for class com.package.main.SetUInt is "SetUInt()", as the default constructor, and "SetUInt(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the value to be written to the PLC; getValues() for returning the values to be written to the PLC; getValues(int) for returning a specific element of the array of values to be written to the PLC; getValue(int) for setting a single value to be written to the PLC; getValues(int, int) for setting a specific element of the array of values to be written to the PLC; and getValues(int[]) for setting the value to be written to the PLC.

Class com.package.main.SetUShort has a public class of "SetUShort", extends SetRef, and implements Serializable. In particular, the SetUShort class is a Bean that is used to write a variable or register(s) reference 'on demand'. Each reference is treated as an unsigned, 8-bit integer. If the address specified for the data item is a variable name, the quantity of references is one (the default), and the variable's data type is USHORT.

In this embodiment, the constructor for class com.package.main.SetUShort is "SetUShort()", as the default constructor, and "SetUShort(ClientHandlerInterface)" as the constructor to set the "comm server" to be used for processing requests. Moreover, the preferred methods are: getValue() for returning the value to be written to the PLC; getValues() for returning the values to be written to the PLC; getValues(int) for returning a

specific element of the array of values to be written to the PLC; getValue(short) for setting a single value to be written to the PLC; getValues(int, short) for setting a specific element of the array of values to be written to the PLC; and getValues(short[]) for setting the values to be written to the PLC.

Package com.package.namespace includes classes: com.package.namespace.VarLookupInterface; com.package.namespace.VerInfo; and com.package.main.ConnectPLC.

Class com.package.namespace.VarLookupInterface has a public interface of "VarLookupInterface" and includes the methods of: get(String); getSymbols(); getVerInfo(); init(InetAddress); and isReadOnly(String, int).

Class com.package.namespace.VerInfo has a public class of "VerInfo", extends Object, and implements Serializable. In this embodiment, the constructor for class com.package.namespace.VerInfo is: "VerInfo()", "VerInfo(String)", "VerInfo(String, long, int)", and "VerInfo(String, String). Moreover, the preferred methods are: equals(Object); readData(BufferedReader) for reading delimited data from a file; toString(); and writeData(PrintWriter) for writing delimited data to a file.

Package com.package.vars includes the class com.package.vars.VarInfo having a public class of "VarInfo", extending Object, and implementing Serializable. The constructors include VarInfo() and VarInfo(String, String, int, int, short, short, short, boolean. The methods include: getAddress(); getDataType(); getName(); getOffset(); getSymbolID(); getSymbolType(); getVariableType(); IsReadOnly(); readData(BufferedReader) for reading delimited data from a file; setAddress(String); setDataType(short); setName(String); setOffset(int); setReadOnly(boolean); setSymbolID(int); setSymbolType(short); setVariableType(short); and writeData(PrintWriter) for writing delimited data to a file.

While the specific embodiments have been illustrated and described, numerous modifications come to mind without significantly departing from the spirit of the invention and the scope of protection is only limited by the scope of the accompanying Claims.